Interactive Simulations and advanced Visualization with Modelica

Tobias Bellmann
Institute of Robotics and Mechatronics, German Aerospace Center (DLR)
Münchner Straße 20, 82234 Weßling, Telefon: +49 8153 28-0, Fax: +49 8153 28-2243

Abstract

In this paper a Modelica library for interactive simulation and advanced visualization called ExternalDevices is introduced and presented. Providing support for standard input devices like keyboard and joystick as well as for communication via UDP and shared memory, this library allows the user to interact with a running simulation and process the output data of the simulation in other processes capable of UDP connections. An advanced visualization system replaces the standard Dymola visualization and offers additional features like full-screen viewing, transparency and support for flexible bodies.

Keywords: interactive simulation; visualization; simulation; network; flexible bodies

Introduction

Simulations with Modelica normally are not designed for interactive control. In the standard Modelica 3.1 library, no blocks for input devices or other control possibilities are existent. Nevertheless it can be help- and useful to interact with a running multi-physics simulation, either to reduce the effort needed for the generation of input data for the simulation, or to react directly to the results of a running simulation.
The integrated visualization of the Modelica MultiBody Library is vendor-specific. It is therefore limited to the specified visualization methods provided by the simulation tool. The visualization definitions of Modelica 3.1 are limited to some basic features like some elementary shapes and untextured .dxf CAD files.
To overcome the missing interaction and visualization possibilities, the *ExternalDevices* Library provides a set of blocks and techniques to allow interactive simulations, as well as an advanced real-time visualization of the running simulation, considerably extending the scope of operation especially for multi-body simulations. The *ExternalDevices* library is structured in the following functional packages:

- Input devices: Blocks for the direct control of simulation states by the user
- Communication devices: Blocks allowing the simulation to communicate with other processes via network or shared memory
- External visualization: Blocks and models replacing vendor-specific visualization systems and adding additional visualization possibilities.

The *ExternalDevices* library links either to static or dynamic C++ libraries to provide this additional functionalities. It is available for Dymola 7.x, in a version for Windows, a version for Unix/Linux is planned. Every Modelica implementation able to link external C libraries can use the *ExternalDevices* library and the visualization system.

1 Input devices

Input devices are needed for interactive control of the simulation states, for example to trigger events or to control actors of a multi-body simulation. For this purpose the ExternalDevices library provides blocks for three common PC input devices:

Keyboard: This block allows the monitoring of single keyboard keys, and has a boolean output for the chosen key state. Several blocks can be used parallel, each with a selectable key ID like VK_Return for the return key.

Figure 1: Input devices blocks

Joystick: This block includes support for three axis, eight buttons joysticks for Modelica. The joysticks must be configured and calibrated correctly in the Windows system control panel. Several blocks at the same time are usable with separate joystick IDs, allowing the use of more than one joystick attached to the PC. The three outputs are from the type Real and are normed from -1 to 1 for each joystick axis.

SpaceMouse: The SpaceMouse block includes support for the 3dConnexion Spacemouse, a six DOF input device originally developed at the Institute of Robotics and Mechatronics of the german aerospace center (DLR). This devices consists of a pressure-sensible handle, which can be pushed and rotated to manipulate objects in three-dimensional space. The output connectors of this block also are normed from -1 to 1 and all buttons of the Spacemouse are retrievable via a boolean output vector.

2 Communication devices

It is often useful to control a simulation via network or to process the simulation output data in another program or simulation. The *ExternalDevices* library supplies the user with blocks to communicate with external processes via UDP or Shared Memory. As an example, one simulation can provide input data for another simulation via UDP.

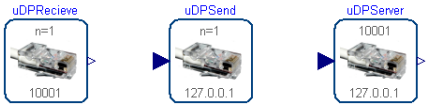

Figure 2: Network devices blocks

UDPRecieve block: This block introduces an UDP client communication input for Modelica. The incoming data must be binary coded double vectors, in the form [double 1,.....,double n] with the selectable length n. The listening port can be selected and must be unique on the system. More than one block in a model may be used under this premise.

UDPSend block: This block allows the sending of Real vectors with selectable length n. Parameters are the target port and IP address. More than one block may be used in a model.

UDPServer block: Combination of UDPSend and UDPRecieve functionality.

Shared memory Block: A shared memory block, using the QT framework from QT Software [1] allows the communication between two processes on the same computer system. Supporting Real, Integer and Boolean vectors it can be used as an interface to other parallel running simulations or processes with the same QT shared memory interface. Several blocks can be used within the same model by defining different memory storage IDs.

3 A model-based approach for visualization

Including the complete Modelica 3.0 standard for visualization of multi-body models, this library furthermore allows the user to build more complex visual environments within the Modelica model, to be simulated on an external visualization tool, DLR SimVis. The visualization package uses network communication to transmit the visualization data from the simulation to the external visualization tool. This tool, DLR SimVis, provided together with this library is based on OpenGL [4] and the OpenSceneGraph [2] 3D scenegraph.
The main purpose is the visualization of multi-body simulations, replacing vendor-specifig graphic engines with a presentable graphic engine, supporting full-screen viewing and a large variety of textured 3D CAD file formats.
Every object visualized in the external viewer is provided by a visualization component in Modelica, containing all necessary data for correct visual representation.

3.1 Object-oriented approach for visualization components

The object-oriented approach of the *ExternalDevices* visualization system easily allows to integrate the visualization components in to physical components. The complete informations necessary for the visualization are already existent in the physical component and can be used in the visualization block. Figure 3 shows the integration of a visualization block in the model of a spring. The information about the spring's position, orientation and length is available via the frame connectors, other parameters like winding number, wire diameter e.g. can be chosen via parameter dialog.

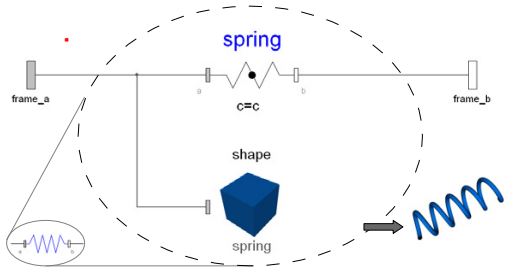

Figure 3: Integration of the visualization component in a physical model

3.2 Comparison with existing physical visualization systems

Most existing visualization systems rely on a central configuration file defining the scene and the input channels for a specific visualization task. The MathWorks VR Toolbox [5] for example uses VRML Files, containing the complete kinematic and degrees of freedom for scene manipulations as well as the CAD Data. In the corresponding Simulink model, the VR Block provides the inputs for the desired degrees of freedom in the scene. These inputs are now connected with the according signals of the Simulink physics model. This leads to increased configuration efforts if the model has to be changed, because both the VRML Configuration and the signals have to be adapted. Another example for the separation of model and visualization system is the VisEngine of Aerolabs GmbH [6]. The VisEngine uses a configuration file defining the used CAD data for the scene and the input channels (e.g. UDP, tables from the file system, etc.) for moving the CAD objects.

In both cases, a change in the model requires a modification in the visualization. With the object-oriented approach of *ExternalDevices* combined with Modelica, this is not necessary, because of the complete integration of the visualization into the model components.
Because of an additional software layer, hidden from the user, there is no need for complicated additional signal connections. The visualization data is collected in a data core controlled by an *ExternalObject* construction and transmitted to the external visualization viewer (see Figure 4).

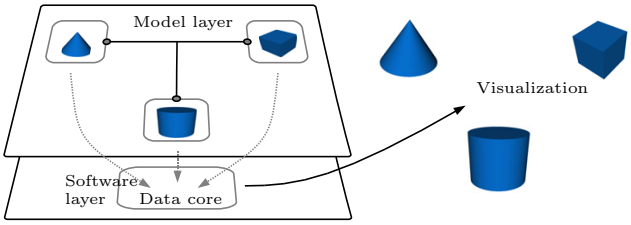

Figure 4: Model layer and software layer with data administration (hidden)

3.3 Modified Modelica 3.0 standard library

One strength of Dymola's visualization system is the automatic generation of the scene via the visualization properties of every multi-body system. This is done via implementing a visualization definition in every part of the Multi-body library. These definitions all inherit the *Modelica.Mechanics.MultiBody.Visualizers.Advanced.Shape* block, which is the connection to the Dymola visualization. By replacing this block with a modified variant, the complete model visualization is redirected to the external visualization. This modification allows to use every existing model with the external Visualization.
Since Modelica 3.1 the vendor-specific library elements like the Shape block are grouped in an additional service library *ModelicaServices*. Analogue to the modified Modelica library, it is possible by providing a customized *ModelicaServices* implementation to redirect the visualization data to the external viewer DLR SimVis.

3.4 External viewer software DLR SimVis

The visualization data is sent from the simulation process via network to the external viewer software SimVis. This software is responsible for the interpretation of the visualization data and for the rendering of the scene. Because of the use of a network communication, the Viewer software does not have to run on the same system as the simulation, and therefore more computing power for the simulation can be provided. Based on the open source scene graph OpenSceneGraph [2] a wide range of formats for CAD data is supported. The continuing development of OpenSceneGraph provides the base for highly detailed visualizations. Utilizing the ffmpeg video en/decoding library [3], several video codecs for video grabbing are supported. The following features already are implemented:

- Fullscreen mode
- Support for multiple cameras
- Multi-monitor support
- Textured CAD files (.dxf, .stl, .3ds, .obj, ...)
- Video grabbing, formats: (MPEG4, MS MPEG4 2/3 (.avi), Flash video (.flv), Huffman Encoding (lossless), Windows Media Video (.wmv))
- Video grabbing with free configurable bit rate and replay speed
- Wireframe mode, Stereo mode (anaglyph and indirectly by OpenGL graphics card drivers)
- Precise replay controls including a jog-dial

3.5 Object-oriented network protocol

In order to reduce the amount of visualization data to be transported over the network connection, an optimized protocol is necessary. In the implementation of the *ExternalDevices* library, an object-oriented approach with data and packet objects has been chosen. While the data objects handle the data storage and the tasks of serialization / deserialization, the packet objects are responsible for data object administration. This includes an incremental packaging of data, where only data objects, which changed in the last time step are included in the transmitted data. A lossless transport protocol provided, like TCP/IP, this method reduces the needed communication bandwidth significantly, because static properties of visualization elements must not be communicated every time step.
Every packet is identified and assigned to a visualization element by an unique, automatically generated ID. Figure 5 provides an overview of the network communication architecture.

4 Visualization package content

The visualization package is structured in subpackages for Shape blocks, Camera blocks, Light blocks, Energy Flow blocks and Effect blocks.

4.1 The *UpdateVisualization* block

The *UpdateVisualization* block is responsible for the integration of the visualization blocks in the simulation process. Similar to the *MultiBody.World* block, it will be automatically inserted as an *inner* component if a visualization block is used in the model. The *UpdateVisualization* block controls parameters like the IP address and port for the communication with the visualization software SimVis and can be used to disable the complete visualization.

During a simulation run, the block triggers a time event every time visualization data shall be sent to SimVis. This update interval time can be varied via a parameter and should be < 0.04 s for 25 frames per second during real-time simulations. If the update time event is triggered, every block sends its data to the data core triggered by the Boolean variable *UpdateVisualization.send*.

4.2 Shape blocks

Every Shape block has a frame connector as input. The resulting forces and torques of such a block is zero, so the block only has visual and no dynamic or kinematic effects. This blocks can be used as

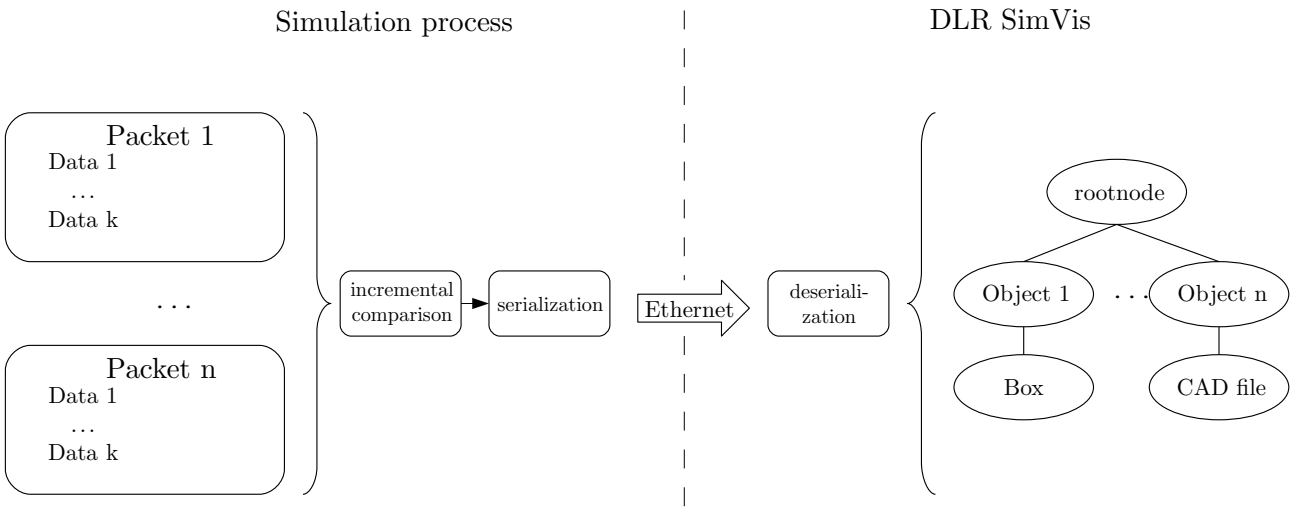

Figure 5: Basic principle of visualization network architecture

integrated components in a Modelica multi-body model:

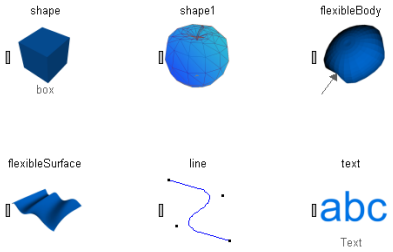

Figure 6: Shape blocks

Elementary Shape: This block represents the supported basic shapes, such as boxes, spheres etc. The available primitives are:

Basic shape type	Existent in Modelica 3.0	Existent in library
Box	Yes	Yes
Sphere	Yes	Yes
Cone	Yes	Yes + Features
Spring	Yes	Yes
Cylinder	Yes	Yes
Pipe	Yes	Yes
Beam	Yes	Yes
Gearwheel	Yes	Yes + Features
Coordinate System	Yes	Yes
Grid	No	Yes

Every *ElementaryShape* can be parametrized in size, color, transparency and reflection behavior (for specular highlights). For the more complex shapes like spring, gearwheel and cone, additional parameters can be set. Beyond the standard parameters, required by the Modelica 3.1 standard, some additional parametrization is possible, for example the operating angle of gearwheels allows the construction of bevel gears.

FileShape: The *FileShape* block allows to use 3D CAD models, supporting numerous file formats like .obj, .dxf, .3ds, .stl, and every other file formats supported by the OpenSceneGraph plug-in system. The key features of this block are support for textured 3D models and additional parameters like transparency and a wire-frame modus. The loaded 3D model can be scaled in x,y,z directions.

Line The *Line* block implements linear or Bezier interpolated lines, with n control points relative to the frame connector of the block.

Text Shape: This block allows to place 2D texts in the scene, aligned to a specified direction or to the screen. Font, character size and color are parameterizable.

Text Shape with Value: In addition to the *TextShape*, this block has a Real input connector. The text in the visualization is followed by the

input value of this connector, allowing the display of simulation data in the visualization.

4.3 Visualization of flexible bodies

Especially for the use in the DLR *FlexibleBodies* library, a visualization module for flexible bodies is available in the library. For topologically simple objects (e.g. beams or tori) a parameterizable surface can be used, visualizing an array of points (see Figure 7)

For more complex models, an spatial interpolating algorithm is implemented. This algorithm allows to deform CAD models visually according to a displacement set of n control points ($n << n_{CAD}$) and interpolates the CAD data points spatially between the control points. With this algorithm the number of points to be communicated to the visualization system can be reduced drastically, as only the displacement set has to be calculated and submitted to the visualization (see Figure 8).

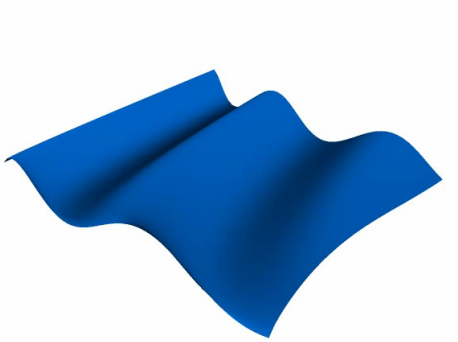

Figure 7: quadratic, parameterizable surface

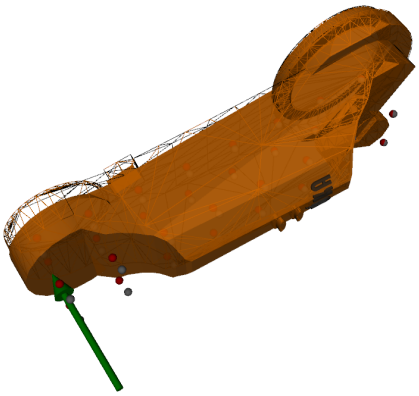

Figure 8: Deformed(wireframe) component of an industrial robot

4.4 Energy flow visualization

The visualization package supports the visualization of energy flows with several blocks (see Figure 9). The energy flow is represented visually by a transparent pipe (can be deactivated) with moving arrows inside. The spatial configuration of the pipe is specified within the model by parameterizing the single components of the pipe (MultiBody Library compatible) with informations like length, diameter, radius of curved segments etc. The basic flow elements available are *StraightPipe*, *CurvedPipe* and *FlexiblePipe* (flexible interpolated). For visualization purposes, the color, size and speed of the arrows can be dynamically changed during the simulation.

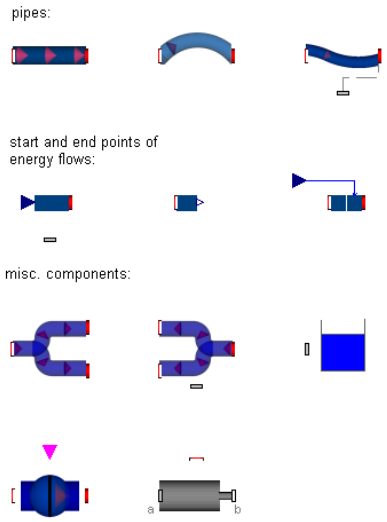

Figure 9: Energy flow blocks

Every energy flow pipe has to begin with a *FlowBegin block* and ends with an *FlowEnd* block. The start position of a pipe is defined by a MultiBody frame connector, whereas the flow speed of the pipe indicators can be set by an Real input of the FlowBegin block. The interconnection between the pipe segments is handled by a special connector containing the connecting frame of the segment, the flow through the pipe and an ID of both connected pipe segments. The IDs are necessary to provide information about the assembly of the energy flow system for the visualization software as a double-linked list.

In order to avoid kinematic loops, the frame information of the pipe segments is encapsulated

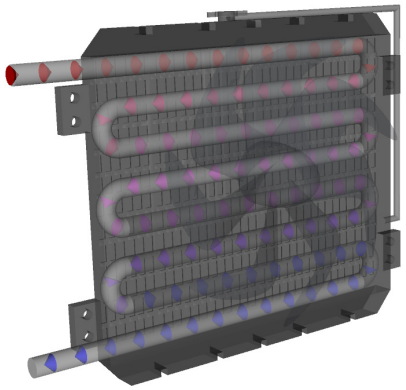

Figure 10: Car Radiator with coolant flow

in the connector and should not be accessed directly. If a connection between two defined points is desired, the *FlexiblePipe* block can be used to exactly construct a pipe connecting these two points. This is done by a Bezier interpolation algorithm and can be updated during the simulation to visualize a flexible connection between two moving points.

4.5 HUD Elements:

This objects support the generation of simple head-up-displays, allowing the placement of text, bar graphs and bitmaps. This elements can be combined to form e.g. analogue instruments (see Figure 11) or digital gauges. The displayed values are updated during the simulation and allow a direct insight into the connected states of the simulation. The head-up-display is placed as a 2D overlay over the 3D scene.

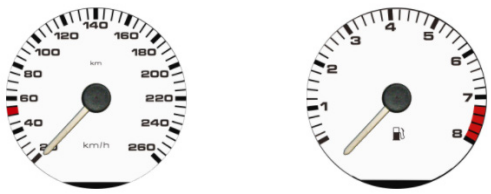

Figure 11: Several bitmaps combined to an analogue tachometer and rpm gauge

4.6 Cameras

The visualization system supports multiple camera views. If there is no dedicated camera in the model, a standard view will be used. For every new camera block in the model, an additional subwindow is available in the visualization viewer, showing the scene from the cameras perspective. Every camera can have its own background color, viewing distance and field of view. A full screen mode allows to display the camera perspective on the complete display, multiple displays are supported. The following camera blocks are available:

FreeCamera: A free movable camera, initialized at the camera-frame connectors start position. The cameras position and perspective can be adjusted in the viewer with the computer mouse.

FixedCamera: The perspective and position of this camera is fixed. The position of the camera only can be changed by the simulation itself, and no user interaction is possible. The direction of the camera view can be parametrized.

FollowCamera: This camera is centered on the position of the camera's reference-frame connector. As the reference connector moves, the camera keeps focused on this position. The position of the camera is defined via the camera frame connector.

DynamicFollowCamera: The position of this camera is defined, like the *FollowCamera* via the camera frame connector position. But unlike the Follow Camera it is no direct coupling but a delayed following behavior characterized by a PT 1 system. The time constant of this following behavior can be parametrized.

AttachedCamera: A camera with free adjustable perspective, but with a position defined by the camera's reference-frame connector. This camera is useful for the observation of a dedicated object from different perspectives.

4.7 Lights

To create a well lightened scene, this sub-package provides Light blocks. Without a dedicated lighting block in the model, a standard light will be created and placed at the position of the camera.

Figure 12: Lighting blocks

Light block: This is the most flexible lighting block, including the complete OpenGL definition for lights. Ambient, specular and diffuse light colors as well as directional lighting and attenuation of light are parameterizable.

Spotlight block: To reduce the configuration effort, this block provides a preconfigured spotlight with a selectable color and a spot angle and direction. The specular and diffuse color are set to the same value as the light color.

Diffuse Light block: This block provides a preconfigured directional light, with selectable color and light direction. This light can be used for environment lighting, as it produces parallel light rays like sunlight.

4.8 Effects

This sub-package contains blocks for additional visual effects. In the current release version of the *ExternalDevices* library, the available effects are weather and particle effects.

Weather effect block: Especially for environment visualization, this block provides the three effects fog, rain and snow for more realism. The fog effect, combined with a reduced viewing distance of the camera can be used to reduce the viewing distance in the scene, in order to increase the frame rate of the visualization.
The Rain and Snow effect are done with a particle system. A wind strength can be parametrized. All weather effects can be triggered with a boolean input by the simulation. Weather effects have no distinct position but are located around every camera.

Particle effect block: With this block simulation of smoke or fire with variable intensity is possible. A wind strength can be parametrized as well as particle size and -lifetime. The ParticleEffect block can be connected to MultiBody systems with a frame connector defining the position of the particle origin.

5 Application examples

The following samples are generated with the ExternalDevices library and are demonstrating selected models with external visualization.

Example 1: Electric motor Figure 13 shows a electrical engine propelling a rotor. In Figure 14 the associated Modelica model is shown, with a magnification of the motor. File Shape blocks are used to represent the CAD Data of the motor and the rotor, a Flow Shape block visualizes the energy flow between motor and rotor.

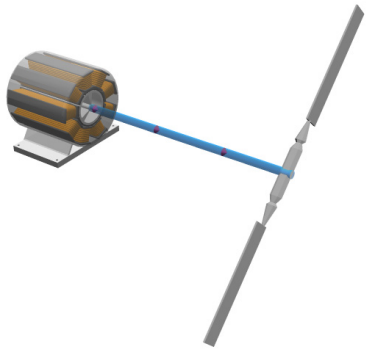

Figure 13: Electrical motor example

Example 2: Hybrid vehicle Figure 15 shows a screen shot of a visualization of a hybrid vehicle, rendered with SimVis. Advanced rendering effects like the transparency of the chassis allow an insight into the car's components. The simulation is controlled with a SensoDrive steering wheel via CAN bus, and a Logitech pedal system via USB joystick input. In Figure 16 the complete driving simulation, as shown on the FISITA world automotive conference 2008 in Munich can be seen. There are actually two simulations running, a driving simulation and a simulation of a robot based motion simulation (see also Example 3). The driving simulation renders two camera perspectives on the left and upper display, while the motion simulator is shown on the right display. The motion simulator receives acceleration

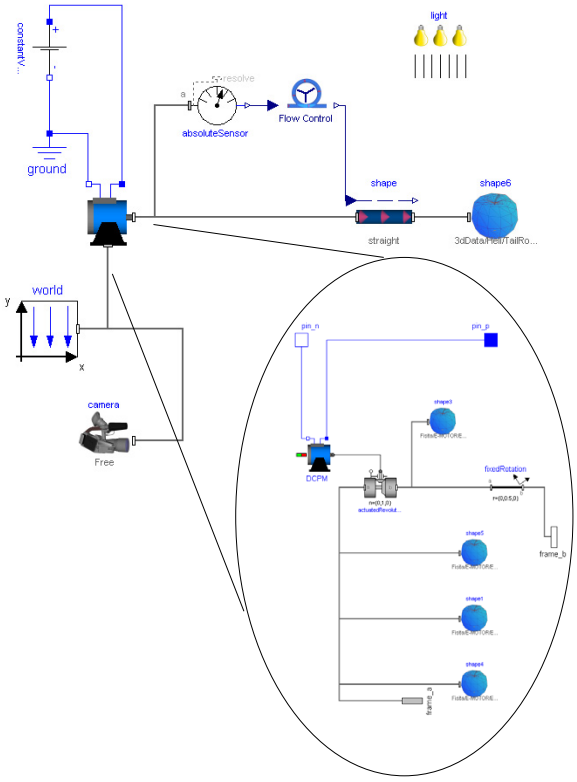

Figure 14: Model of the example

and angular velocity data from the driving simulation and creates a trajectory simulating these motions.

Example 3: Robot visualization Figure 17 shows a visualization of a KUKA KR500/1 robot (Source CAD Data: kuka.com) used as motion simulator. The effects of specular highlights are visible, generating a more plastic look of the robot. In this simulation, the robot joint angles are received via UDP from the real KUKA KR500/1's control computer, the path-planning of the motion simulator is done in a Modelica model and transmitted back to the robot control.

5.1 General performance

During internal tests and projects, scenes with 400 dynamically moved objects have been created, reaching frame rates >25 fps. Models with a memory sizes up to 600 MB (uncompressed) have been loaded and used as scenery. With the increasing rendering power of actual graphic processing units, CAD models with numbers of vertices $> 10^6$ can be displayed with acceptable frame rates.

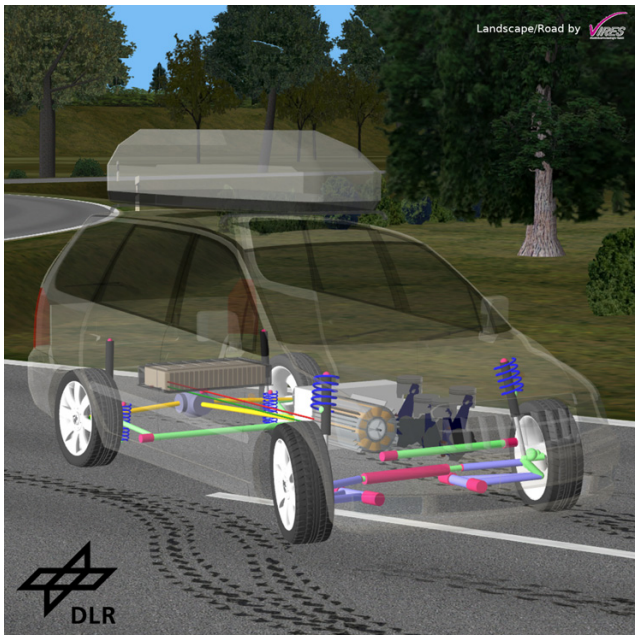

Figure 15: Visualisation of a hybrid vehicle

Figure 16: Driving simulator shown at FISITA 2008

6 Conclusion and Outlook

The *ExternalDevices* library has shown its usefulness during DLR internal tests, especially for creating dynamic, interactive simulations. The possibility to interact with the simulation reduces the effort to generate input trajectories and allows the user to determine the progress of the simulation.

Figure 17: Visualisation of KUKA KR500/1 industrial robot

With UDP and shared memory interfaces, the cooperation of several simulations or reading and writing to other external sources is feasible.
The complete replacement of vendor-specific visualizations and the shift to a platform independent set of Modelica blocks allows much more flexibility in the development of visualization solutions.
The further development of the library will now focus to extend the support of input instruments (e.g. six-axis, force-feedback Joysticks) and the improvement of the visualization, with a new, modular HUD system and full integration of particle systems. Another very focused project aims to integrate plug-ins for loading large terrain databases into the SimVis framework.

References

[1] QT Software: `http://www.qtsoftware.com/products/`

[2] OpenSceneGraph: `http://www.openscenegraph.org/`

[3] FFMEPG: `http://ffmpeg.org/`

[4] OpenGL - The Industry Standard for High Performance Graphics: `http://www.opengl.org/`

[5] Simulink 3D Animation 5 - User's Guide, MathWorks Inc, `http://www.mathworks.de/access/helpdesk/help/pdf_doc/sl3d/sl3d.pdf`

[6] Aerolabs VisEngine: `http://www.aerolabs.net`

[7] Vires Simulationstechnik GmbH: `http://www.vires.com/`